Bilgisayar Bilimleri ve Mühendisliği Dergisi

Cilt-19
Sayı-1
2026

Yapay Zeka Destekli Üretilen Kodlar ile İnsan Yazımı Kodların Karşılaştırılması: Python Örneği

Comparing AI-Assisted Code with Human-Written Code: A Python Case Study

Seda EYGÜ
Atatürk Üniversitesi
Sosyal Bilimler Enstitüsü
Yönetim Bilişim Sistemleri Anabilim Dalı
Erzurum, Türkiye
seda.eygu@atauni.edu.tr
ORCID: 0000-0002-1134-8120

Mustafa KESKİNKILIÇ
Atatürk Üniversitesi
İktisadi ve İdari Bilimler Fakültesi
Yönetim Bilişim Sistemleri Bölümü
Erzurum, Türkiye
muskes@atauni.edu.tr
ORCID: 0000-0002-3394-5575

Öz

Yapay zeka (YZ) temelli kod üretim araçlarının yükselişi, yazılım mühendisliğinde kod yazım süreçlerini dönüştürmektedir. Bu çalışma, YZ destekli "Bolt.New" aracıyla üretilen Python programlarını, insan eliyle yazılmış eşdeğer programlarla kalite, karmaşıklık ve okunabilirlik açısından sistematik olarak karşılaştırmayı amaçlamaktadır. 2019 öncesi GitHub'dan seçilen 30 adet insan yapımı program ile aynı işlevleri gerçekleştiren YZ üretimi programlar, satır sayılarına göre basit, orta ve karmaşık ulamlara ayrılarak incelenmiştir. Python'a özgü statik inceleme aracı Pylint ile ölçülen kalite puanı, hata sayısı, siklomatik karmaşıklık, stil ihlali, kod tekrar oranı ve belge puanı ölçüleri, iki küme arasında istatistiksel olarak karşılaştırılmıştır. Bulgular, YZ'nin basit görevlerde yüksek kalite ve düşük hata oranı sunduğunu, karmaşık görevlerde ise daha düşük karmaşıklık ve daha iyi belge sağladığını, ancak bağlamsal uygunlukta sınırlamalar gösterebildiğini ortaya koymaktadır. Bu çalışma, YZ destekli kod üretiminin Python'a özgü ölçünlere uyumunu ve hata azaltma potansiyelini nesnel ölçülerle değerlendirerek, yazılım geliştirme süreçlerinde otomasyon ve insan denetimi arasında denge kurulmasına yönelik rehber bir çerçeve sunmaktadır.

Makale Bilgileri	**Article Info**
Türü: Araştırma	Type: Research
Geliş tarihi: 10.09.2025	Received date: 10.09.2025
Kabul tarihi:26.11.2025	Accepted date: 26.11.2025

Atıf/ to Cite (IEEE): S. EYGÜ, M. KESKİNKILIÇ, Yapay Zeka Destekli Üretilen Kodlar ile İnsan Yazımı Kodların Karşılaştırılması: Python Örneği: Bilgisayar Bilimleri ve Mühendisliği Dergisi, Cilt 19 2026 Sayı-19-1. Sf: 24-37,
DOI: 10,54525/bbmd.1781375

Anahtar Sözcükler: Yapay Zeka, Kod Kalitesi, Python, Yazılım Mühendisliği, Kod Okunabilirliği, Kod Karmaşıklığı
JEL Sınıflandırması: C88, L86, O33

Abstract

The rise of artificial intelligence (AI)-based code generation tools is transforming code writing processes in software engineering. This study aims to systematically compare Python programs generated by the AI-powered "Bolt.New" tool with equivalent human-written programs in terms of quality, complexity, and readability. Thirty human-written programs selected from GitHub prior to 2019, along with AI-generated programs performing the same functions, were categorized into simple, medium, and complex based on line count and analyzed. Metrics such as quality score, error count, cyclomatic complexity, style violations, code duplication rate, and documentation score, measured using the Python-specific static analysis tool Pylint, were statistically compared between the two groups. The findings reveal that AI offers high quality and low error rates in simple tasks, while in complex tasks, it provides lower complexity and better documentation but shows limitations in contextual appropriateness. This study evaluates the compliance of AI-generated code with Python-specific standards and its potential for error reduction using objective metrics, offering a guiding framework for balancing automation and human oversight in software development processes.

Keywords: Artificial Intelligence, Code Quality, Python, Software Engineering, Code Readability, Code Complexity, JEL Classification: C88, L86, O33

1. Giriş

Yazılım mühendisliği, teknolojinin hızla geliştiği bu bilişim çağında hem akademik hem de diğer uygulamalarda önemli bir rol üstlenmektedir. Bu disiplinin temel yapı taşlarından biri olan kod üretim süreci, yıllar boyunca insan yaratıcılığı, bilgi birikimi ve mühendislik sezgilerine dayalı olarak gelişmiş ve olgunlaşmıştır. 20, yüzyılın ortalarından itibaren geliştirilen yapılandırılmış programlama, nesne yönelimli tasarım ve modüler yazılım geliştirme yaklaşımları, kodun kalite, sürdürülebilirlik ve anlaşılabilirlik boyutlarında sistematik iyileştirmeler yapılmasına olanak tanımıştır. Bu çerçevede, yazılım kalitesi; yalnızca kodun doğru çalışmasıyla sınırlı olmayan, okunabilirlik, bakım kolaylığı, modülerlik ve hata yönetimi gibi çok boyutlu metriklerle tanımlanan bir kavram haline gelmiştir.

Ancak 2010'ların ortalarından itibaren yapay zeka (Yapay Zeka: YZ, Artificial Intelligence: YZ) teknolojilerinde yaşanan atılımlar, yazılım üretimi bakış açısında köklü bir dönüşüm başlatmıştır [13]. Özellikle Doğal Dil İşleme (DDİ), Büyük Dil Modelleri (BDM) ve derin öğrenme sistemlerinin yaygınlaşmasıyla ortaya çıkan üretken yapay zeka, geliştiricilerin kod yazma süreçlerine yardımcı olan ya da doğrudan işlevsel kodlar üretebilen YZ destekli sistemler ortaya çıkmıştır [29]. GitHub Copilot, Codex ve Bolt.New gibi araçlar, yalnızca doğal dil açıklamalarına dayanarak Python, JavaScript, Java gibi dillerde çalışabilir yazılım bileşenleri oluşturabilmekte ve bu sayede geliştiricilerin verimliliğini artırmakta, rutin görevleri otomatikleştirmekte önemli roller üstlenmektedir [9, 17].

Bu gelişmeler, yazılım mühendisliğinde yalnızca üretim hızının değil, aynı zamanda üretim kalitesinin de YZ araçlarıyla nasıl evrileceğine dair yeni soruları gündeme getirmiştir. Kaynaklarda, YZ tarafından üretilen kodların bağlamı doğru yorumlama becerisi, stil bütünlüğü, hata olasılığı ve yapısal karmaşıklık gibi açılardan insan eliyle yazılmış kodlardan farklılık gösterdiği yönünde bulgular yer almaktadır. Özellikle Pearce ve arkadaşları [21], Copilot gibi araçların bağlamdan kopuk ve güvenlik açıklarına açık kod parçaları üretebildiğini belirtmiştir. Nguyen ve arkadaşları [20] ise YZ destekli kodların stil kurallarına daha az uyduğunu, ancak bu durumun statik inceleme araçları ile telafi edilebilir olduğunu öne sürmüştür. Buna karşın, bazı çalışmalar YZ kodlarının daha sade, modüler ve okunabilir olabileceğini savunmakta; bu da alanda çelişkili bulguların bulunduğunu göstermektedir.

Bu bağlamda, bu çalışmanın temel amacı, YZ destekli kod üretim araçlarının yazılım mühendisliği açısından kalite, okunabilirlik ve karmaşıklık gibi ölçüler üzerinden insan üretimi kodlarla sistematik olarak karşılaştırılmasıdır. Bu amaçla, Python dilinde yazılmış 30 adet insan yapımı program ile Bolt.New adlı bir YZ aracı tarafından aynı görevler için oluşturulmuş 30 eşdeğer program incelenmiştir. Programlar, satır sayılarına göre basit, orta ve karmaşık olmak üzere üç ulama ayrılarak zorluk düzeyi temelinde kümelendirilmiştir. Tüm programlar, Python ortamında yaygın olarak kullanılan statik inceleme aracı Pylint ile incelenmiş; her biri için Pylint kalite değeri, toplam hata sayısı, siklomatik karmaşıklık, stil ihlali sayısı, kod tekrar oranı ve belge puanı gibi ölçüler elde edilmiştir.

Çalışma, yalnızca YZ ve insan kaynaklı kodlar arasında genel bir karşılaştırma yapmakla kalmayıp, bu farkların zorluk düzeyine göre değişip değişmediğini de incelemektedir. Ayrıca, stil uyumu ve okunabilirlik gibi daha öznel değerlendirmelerin Pylint gibi nesnel araçlarla nasıl ölçümlenebileceğini göstermektedir. Bu yönüyle çalışma, yazılım mühendisliği kaynaklarında sıklıkla göz ardı edilen YZ araçlarının nesnel kalite ölçüleri bağlamında sistematik değerlendirilmesini hedeflemekte ve YZ araçlarının yazılım geliştirme süreçlerine entegrasyonu konusuna somut veri temelli katkılar sunmaktadır.

2. Kaynak Taraması

Son yıllarda geleneksel insan yazımı kodlar ile YZ destekli kod üretim araçlarının karşılaştırılması, çeşitli ölçütler üzerinden yapılan birçok çalışma ile geniş bir kaynak oluşturmuştur. Bu çalışmalar, kod üretim süreçlerinin evrimini anlamak ve yeni teknolojilerin yazılım geliştirme üzerindeki etkilerini değerlendirmek açısından önemli veriler sunmaktadır. Bu çerçevede, kod üretiminin tarihsel gelişimini ve kavramsal temellerini ele almak, kaynaklardaki tartışmaları bütüncül bir yaklaşımla değerlendirmek açısından önemlidir.

2.1 Yazılım Mühendisliğinde Kod Üretimi: Tarihsel ve Kavramsal Arka Plan

Yazılım mühendisliği, bilgi teknolojilerindeki ilerlemelerle birlikte sürekli dönüşen bir alan olarak, kod üretimi süreçlerinde de köklü değişimlere tanıklık etmiştir. Kod yazımı, yazılım geliştirme yaşam döngüsünün temel yapı taşlarından biri olup, geleneksel insan merkezli yaklaşımlar kadar, günümüzde yaygınlaşan otomasyon araçlarıyla da gerçekleştirilir. Bu bölümde, yazılım mühendisliğinde kod üretiminin tarihsel süreci ve kavramsal temelleri açıklanarak; kod kalitesi, okunabilirlik ve karmaşıklık gibi önemli metriklerin önemi vurgulanmakta ve klasik yöntemlerle yapay zeka destekli kod üretimi arasındaki farkların incelenmesine zemin hazırlanmaktadır.

Yazılım mühendisliğinin ilk dönemlerinde, kod yazımı tamamen geliştiricilerin bilgi ve deneyimine dayalı bir süreçti. 1960'lar ve 1970'lerde yapılandırılmış programlama yaklaşımlarının geliştirilmesi, kodun daha planlı ve sistematik yazılmasına katkı sağladı [11]. Bu dönemde yazılım kalitesini değerlendirmek amacıyla ilk ölçüler ortaya çıkmaya başladı. Örneğin, Boehm ve arkadaşları [4], yazılımın güvenilirliği ile bakım kolaylığı gibi kavramları tanımlayarak kalite değerlendirmesine kuramsal bir çerçeve kazandırdı. Boehm'un bu çalışması, yalnızca işlevsellik değil; aynı zamanda kodun okunabilirliği ve sürdürülebilirliği gibi unsurların da kaliteyi belirleyen öğeler olduğunu vurgulamıştır [4]. Bu değerlendirme ölçütleri, hem geleneksel hem de YZ tabanlı kodlar için günümüzde hâlen geçerliliğini korumaktadır.

1980'li yıllardan itibaren otomasyonun yazılım geliştirme sürecine dahil olması, kod üretimini yeni bir boyuta taşıdı.

Derleyiciler, hata ayıklama araçları ve entegre geliştirme ortamları (IDE'ler), geliştiricilerin daha sistematik çalışmasına olanak tanıdı; fakat yine de insan kararları süreçte belirleyici olmaya devam etti [26]. Kod kalitesini değerlendirmek için geliştirilen statik inceleme araçları bu dönemde önem kazandı. Pylint gibi yazılımlar, özellikle Python dilinde kodların yapısını, stilini ve hata olasılıklarını inceleme ederek belirli kalite standartlarının korunmasına yardımcı olmaktadır [23]. Bu tür araçların kullanımı, özellikle açık kaynak yazılım projelerinde kaliteyi izleme ve iyileştirme bakımından kritik önem taşımaktadır [2].

Kod üretiminde bir diğer anahtar kavram olan karmaşıklık, yazılımın anlaşılabilirliği ve bakım süreçlerini doğrudan etkilemektedir. McCabe'in [18] geliştirdiği siklomatik karmaşıklık ölçütü, bir yazılım parçasının kontrol akışını inceleyerek karmaşıklık derecesini ölçmenin etkili bir yolunu sunmuştur. Bu ölçüt, yazılımın yalnızca hata olasılıklarını değil, aynı zamanda test edilebilirlik ve sürdürülebilirlik yönlerini de değerlendirmeye olanak tanımaktadır [18]. Halstead'in [14] önerdiği metrikler ise yazılımın hacmini ve zorluk derecesini değerlendirerek karmaşıklık incelemesine farklı bir katkı sunmuştur. Bu göstergeler, çalışmamızda YZ destekli ve geleneksel yöntemlerle yazılmış kodlar arasında yapılacak karşılaştırmalar için teorik temel sağlamaktadır.

Kodun okunabilirliği ise yazılım mühendisliğinde bir diğer önemli konudur. Bu kavram, yazılımın geliştiriciler tarafından kolayca anlaşılmasını ve bakımının kolaylaşmasını hedefler. Buse ve Weimer [7], kod okunabilirliğini değerlendirmek için otomatik metriklere dayalı bir çerçeve geliştirmiş ve stil, isimlendirme kuralları ile yorum satırlarının bu sürece katkılarını ortaya koymuştur. Özellikle yapay zeka destekli kod üretiminde, okunabilirlik kriteri ayrı bir öneme sahiptir; zira YZ ile üretilen kodlar, bağlama uygunlukta zayıflıklar veya stil tutarsızlıkları gösterebilmektedir [20].

Günümüzde GitHub Copilot ve Bolt.New gibi yapay zeka temelli kod üretim araçlarının yaygınlaşması, yazılım geliştirme sürecinde yeni bir çağ başlatmıştır. Bu araçlar, büyük dil modellerine (LLM'ler) dayanarak doğal dil girdilerini işleyip çalışabilir koda dönüştürmektedir [9]. Bununla birlikte, YZ destekli kodların kalite düzeyi, hata ihtimali ve yapısal karmaşıklığı, insan eliyle yazılmış kodlarla kıyaslandığında çeşitli tartışmaları da beraberinde getirmektedir. Pearce ve çalışma arkadaşları [21], bu tür sistemlerin ürettiği kodlarda güvenlik açıkları ve bağlamdan kopukluk gibi risklerin bulunabileceğini öne sürmüştür. Bu nedenle, çalışmamızda klasik yöntemlerle YZ destekli kod üretimi arasındaki sistematik karşılaştırma ihtiyacı açıkça ortaya konmaktadır.

Sonuç olarak, yazılım mühendisliğinde kod üretimi süreci, zamanla insan merkezli yöntemlerden otomatik sistemlere ve yapay zeka destekli araçlara doğru evrilmiştir. Kodun kalitesi, okunabilirliği ve karmaşıklığı gibi kavramlar, her iki üretim yaklaşımında da değerlendirme ölçütü olmaya devam etmektedir. Bu çalışma, Python kodlarının Pylint analiziyle incelenmesi yoluyla, yapay zeka destekli yazılım üretiminin geleneksel yöntemlerle kıyaslanarak güçlü ve zayıf yönlerinin ortaya konmasını hedeflemektedir. Bu bağlamda, mevcut kaynaklarda tanımlanmış olan kavramlar ve ölçüler, araştırmamızın kuramsal zeminini oluşturmaktadır.

2.2. Yapay Zeka Destekli Yazılım Geliştirme Araçlarının Yükselişi

Son yıllarda yapay zeka tabanlı yazılım geliştirme araçlarının hızla yaygınlaşması, kod üretiminde köklü bir dönüşüme yol açmış ve yazılım mühendisliğinde yeni bir dönemi beraberinde getirmiştir. Özellikle büyük dil modellerinin gelişimiyle birlikte, doğal dil ile yazılım kodu üretimi, kod tamamlama ve hata düzeltme gibi işlemler otomatikleşmiş, geliştiricilere önemli kolaylıklar sağlanmıştır. Bu bölümde, YZ destekli yazılım araçlarının tarihsel gelişiminden teknolojik altyapısına kadar birçok boyut ele alınarak, geleneksel yöntemlerle karşılaştırmalı bir inceleme için kuramsal bir temel oluşturulmuştur. Bu çerçevede, çalışmada kullanılan Bolt.New gibi araçlar üzerinden, YZ destekli kod üretiminin kod kalitesi, hata oranı ve kod karmaşıklığı üzerindeki olası etkileri değerlendirilmektedir.

YZ temelli kod üretiminin ortaya çıkışı, DDİ ve makine öğrenmesindeki ilerlemelere dayanmaktadır. Özellikle 2010'ların ortasında derin öğrenme modellerinde yaşanan gelişmeler ve dönüştürücü mimarisinin tanıtılması [29], hem doğal dili hem de yazılım dillerini anlamada çığır açan gelişmelere olanak tanımıştır. Bu bağlamda, Brown ve arkadaşları [6], büyük dil modellerinin az örnekle öğrenme (few-shot learning) kapasitelerinin, bağlamsal anlamayı geliştirme potansiyeline sahip olduğunu, ancak karmaşık senaryolarda sınırlılıklar gösterebildiğini belirtmiştir. Bu, Bolt.New gibi araçların bağlamsal uygunluk sorunlarının, modelin eğitim verilerindeki genellikten kaynaklanabileceğini göstermektedir. Ayrıca, Weisz ve arkadaşları [28], YZ ve insan geliştiriciler arasındaki iş birliğinin, kod üretiminde tamamlayıcı bir rol oynayarak hem verimliliği artırdığını hem de insan denetiminin önemini koruduğunu vurgulamıştır. Bu teknolojik atılımın sonucunda, 2021 yılında GitHub Copilot gibi araçlar geliştiricilerin kullanımına sunulmuş ve yazılım geliştirme pratiklerine entegre edilmiştir [9]. Copilot, Codex modeli sayesinde, geliştiricilerin yazdığı kodu tamamlayabilmekte veya yalnızca doğal dil açıklamalarıyla işlevsel kodlar önerebilmektedir. Bolt.New ise, Python gibi spesifik programlama dillerine odaklanarak daha niş ve hedefe yönelik çözümler üretmeyi amaçlamaktadır [5]. Bu araçların ortak noktası, büyük ve çeşitli yazılım veri kümeleri (örneğin GitHub depoları) üzerinde eğitilerek bağlamla uyumlu kod önerileri üretebilmeleridir [1].

YZ destekli yazılım araçlarının sunduğu başlıca üstünlükler arasında, geliştirici üretkenliğinin artması ve tekrarlayan görevlerin otomatikleştirilmesi yer almaktadır. Ziegler ve arkadaşlarının [31] yaptığı bir çalışmada, YZ tabanlı kod tamamlama sistemlerinin, yazılım geliştirme süresini %20 ila %30 oranında kısalttığı bildirilmiştir. Ancak, Bird ve arkadaşları [3], YZ destekli kod üretiminin etik sorunlar, örneğin telif hakkı ihlalleri ve geliştirici bağımlılığı gibi riskler taşıyabileceğini belirtmiş, bu araçların uzun vadeli etkilerinin dikkatle incelenmesi gerektiğini vurgulamıştır. Özellikle boilerplate kod yazımı, hata ayıklama ve kod stil

düzenlemeleri gibi zaman alan görevlerde, bu araçlar geliştiricilere ciddi bir zaman kazancı sunmakta ve daha yaratıcı süreçlere odaklanmalarını sağlamaktadır [10].

Bununla birlikte, YZ destekli kod üretiminin bazı sınırlılıkları da mevcuttur. Kaynaklarda yer alan çeşitli bulgular, bu araçların ürettiği kodlarda bağlam dışı hatalar, güvenlik açıkları veya gereksiz karmaşıklık gibi sorunların ortaya çıkabildiğini göstermektedir [21]. Örneğin, GitHub Copilot tarafından üretilen kodların yaklaşık %40'ında güvenlik açığı riski bulunduğu rapor edilmiştir. Bu gibi bulgular, çalışmamızın ilk araştırma sorusunu doğrudan ilgilendirmekte olup, insan ve YZ kaynaklı kodların hata oranları yönünden sistematik karşılaştırılmasını gerekli kılmaktadır. Ayrıca, bu sistemler tarafından üretilen kodlar her zaman geliştiricilerin alışkın olduğu okunabilirlik ve stil standartlarına uygun olmayabilir. Bunun nedeni, modellerin çoğunlukla eğitim aldıkları veri kümelerinde baskın olan kalıpları temel alarak üretim yapmalarıdır [20]. Bu bağlamda, çalışmamızda Bolt.New ile yazdırılan kodların okunabilirliği ve insan yazımı kodlarla kıyaslanması önemli bir inceleme boyutunu oluşturmaktadır.

YZ araçlarının kod karmaşıklığı üzerindeki etkisi ise kaynaklarda tartışmalı bir konu olarak öne çıkmaktadır. Bazı araştırmalar, bu araçların gereğinden fazla satır kod üreterek yazılımı daha karmaşık hale getirebileceğini belirtirken [16], bazıları ise aksine, daha sade ve optimize kodlar üretilebileceğini savunmaktadır [17]. Bu çelişkili bulgular, çalışmamızda Bolt.New tarafından yazdırılan kodlarla insan yazımı kodlar arasında Pylint üzerinden siklomatik karmaşıklık gibi metrikler kullanarak karşılaştırma yapılmasının önemini göstermektedir. Python gibi yüksek seviyeli programlama dillerinde, karmaşıklık düzeyi, yazılımın bakım kolaylığı ve test edilebilirliği açısından büyük önem taşımaktadır [7].

Genel olarak bakıldığında, YZ destekli yazılım araçlarının yazılım geliştirme süreçlerine entegrasyonu halen erken bir evrede değerlendirilmektedir. Akademik çalışmalar çoğunlukla popüler araçlara (örneğin Copilot) odaklanmakta olup, Bolt.New gibi daha yeni araçlara ilişkin deneysel veri henüz sınırlıdır. Ayrıca, insan eliyle yazılmış kodlarla YZ araçlarının üretimleri arasında doğrudan ve sistematik karşılaştırmalar yapan deneysel çalışmalar oldukça azdır [10]. Bu bağlamda, bu çalışma, Bolt.New aracıyla üretilen Python kodlarını, 2019 yılı öncesinde GitHub'da yer alan insan yazımı kodlarla karşılaştırarak, YZ destekli yazılım üretiminin avantaj ve sınırlarını daha açık şekilde değerlendirmeyi hedeflemektedir. Bu bölümde aktarılan kuramsal çerçeve, kod kalitesi, hata oranı ve karmaşıklık gibi boyutlarda yapılacak karşılaştırmalara sağlam bir temel sunmaktadır.

2.3. Kod Kalitesi ve Hata Oranı Üzerine Çalışmalar

Yazılım mühendisliği bağlamında, kod kalitesi; bir yazılımın güvenilir, sürdürülebilir ve işlevsel olma düzeyini belirleyen temel ölçütlerden biridir. Bu bağlamda, hata oranı da kod kalitesinin doğrudan bir göstergesi olarak değerlendirilmekte ve yazılımın doğruluğunu ve bakım kolaylığını etkileyen önemli bir unsur olarak öne çıkmaktadır. Geleneksel geliştirme süreçlerinde, bu iki ölçüt uzun süredir statik inceleme araçları ile değerlendirilmektedir. Ancak yapay zeka temelli kod üretim araçlarının yaygınlaşmasıyla birlikte, bu ölçütlerin yeniden gözden geçirilmesi ve değerlendirme yaklaşımlarının çeşitlendirilmesi gerekliliği ortaya çıkmıştır. Bu bölümde, kaynaklarda kod kalitesi ve hata oranı konularına dair yapılan çalışmalar ele alınmakta ve YZ tarafından üretilmiş olan kodların (örneğin Bolt.New aracılığıyla oluşturulanların), geleneksel yöntemlerle geliştirilen insan yazımı kodlarla nasıl karşılaştırıldığını irdeleyen yöntem ve bulgular tartışılmaktadır. Çalışmamız, Python dilinde yazılmış kodlar üzerinde Pylint incelemesi gerçekleştirerek hata oranlarını karşılaştırmakta ve bu doğrultuda literatüre katkı sunmayı hedeflemektedir.

Kod kalitesi kavramı, yazılım mühendisliğinin başlangıcından bu yana önemini koruyan bir araştırma konusu olmuştur. Örneğin Boehm ve arkadaşları [4], kod kalitesini güvenilirlik, taşınabilirlik ve bakım kolaylığı gibi çoklu boyutlar üzerinden tanımlamış ve bu unsurların sistematik olarak ölçülmesi gerektiğini belirtmiştir. Statik inceleme araçları, bu noktada öne çıkan yöntemlerdendir. Pylint gibi araçlar, Python dilinde yazılmış kodlarda sözdizimi hatalarını, stil uyumsuzluklarını ve potansiyel mantık hatalarını belirlemek amacıyla yaygın olarak kullanılmaktadır [23]. Beller ve arkadaşları [2] tarafından yapılan bir çalışmada, açık kaynak projelerde statik inceleme araçlarının etkin kullanımı incelenmiş ve bu araçların hata oranlarının azaltılmasında etkili olduğu gösterilmiştir. Bu bulgular, çalışmamızda Pylint'in YZ destekli ve geleneksel kodlar arasında kalite karşılaştırması yapmak için uygun bir araç olduğunu göstermektedir.

İnsan tarafından geliştirilen yazılımlarda hata oranlarını etkileyen etkenler arasında geliştiricilerin deneyim düzeyi, projenin karmaşıklığı ve kullanılan test süreçlerinin kalitesi yer almaktadır. İnsan kaynaklı hatalar genellikle bağlama ilişkin yanlış anlamalar, dikkatsizlikler ya da kod stiline dair tutarsızlıklarla ilişkiliyken, YZ temelli araçlarla üretilen kodlarda hata profilleri belirli yönlerden farklılık göstermektedir [10]. Örneğin, Carlini ve Wagner [8], YZ modellerinde hataların veya güvenlik açıklarının, modelin öğrenme sürecindeki özelliklerden kaynaklanabileceğini ve bu hataların bazen sistematik olarak ortaya çıktığını belirtmiştir. Benzer şekilde, Mozannar ve arkadaşları [19], YZ kod üretim modellerinin sağlamlığını değerlendirerek, bu modellerin özellikle bağlamsal hatalara ve nadir görülen senaryolara karşı hassas olabileceğini göstermiştir. Bu, Bolt.New gibi araçların ürettiği kodlarda gözlemlenen bağlamsal hataların, modelin eğitim verilerindeki sınırlılıklarla ilişkili olabileceğini düşündürmektedir. Bu çalışmada, YZ tarafından oluşturulan ve insan eliyle yazılan kodlar arasında bu hata yapılarının nasıl ayrıştığını sistematik olarak incelemek ve YZ'nin hata azaltmadaki potansiyelini ortaya koymak amaçlanmaktadır.

YZ destekli kodlama araçlarının hata oranları üzerine yapılan araştırmalar, bu araçların hem sunduğu olanaklara hem de sahip oldukları sınırlılıklara dikkat çekmektedir. Örneğin Chen ve arkadaşları [9], Codex tabanlı GitHub Copilot'un kod üretiminde yüksek doğruluk oranlarına ulaşabildiğini, ancak zaman zaman bağlamı yanlış yorumladığını ve hatalı mantıksal

akışlar oluşturduğunu ifade etmiştir. Finnie-Ansley ve arkadaşları [12], OpenAI Codex'in özellikle basit programlama görevlerinde insan düzeyinde performansa yakın sonuçlar ürettiğini, ancak bağlamsal derinlik gerektiren durumlarda hata oranlarının artabileceğini göstermiştir. Aynı şekilde Pearce ve arkadaşları [21], YZ tarafından üretilmiş kodların yaklaşık %40'ında güvenlik açığı tespit etmiş ve bu tür kodların doğrudan üretim ortamlarında kullanılmadan önce dikkatle değerlendirilmesi gerektiğini vurgulamıştır.

Hata oranlarının ötesinde, kod kalitesi değerlendirmesinde dikkate alınan diğer önemli unsurlar arasında stil bütünlüğü ve belirli standartlara uygunluk da yer almaktadır. Nguyen ve arkadaşları [20], Python kodlarında stil ihlallerinin (örneğin PEP 8 standartlarına uyumsuzluk) YZ tarafından üretilen kodlarda daha yaygın olduğunu, ancak bu tür hataların otomatik araçlarla büyük ölçüde düzeltilebildiğini belirtmiştir. Bununla birlikte, mantıksal hatalar özellikle karmaşık algoritmalar ya da özel uygulama gereksinimlerine sahip projelerde, YZ tarafından oluşturulan kodlarda daha zor tespit edilebilmektedir [16]. Bu durum, çalışmamızda Pylint analizinin yalnızca söz dizimsel ve stil hatalarını değil, aynı zamanda potansiyel mantık hatalarını da göz önünde bulundurmasının neden önemli olduğunu ortaya koymaktadır.

YZ destekli ve insan kaynaklı kodların hata oranlarının sistematik biçimde karşılaştırıldığı çalışmalar kaynaklarda oldukça sınırlıdır. Örneğin Dakhel ve arkadaşları [10], GitHub Copilot tarafından üretilen kodların hata oranlarını insan geliştiricilerle karşılaştırmış ve basit görevlerde YZ'nin daha az hata ürettiğini, ancak karmaşık görevlerde insan denetiminin gerekli olduğunu ifade etmiştir. Ancak Bolt.New gibi daha güncel ve özelleşmiş YZ araçlarının hata profilleriyle ilgili derinlemesine çalışmalar henüz sınırlıdır. Bu noktada, çalışmamız; 2019 öncesi GitHub verilerinden elde edilen insan yazımı Python kodları ile Bolt.New tarafından oluşturulan kodları Pylint kullanarak inceleme etmekte ve YZ destekli kod üretiminin hata oranı bakımından avantaj ve sınırlamalarını ortaya koymayı amaçlamaktadır. Bu bölümde sunulan literatür, kod kalitesi ve hata oranı analizimiz için teorik bir temel işlevi görmektedir.

2.4. Kod Okunabilirliği ve İnsan-YZ Kodlarının Karşılaştırmalı İncelenmesi

Yazılım mühendisliğinde kodun kolay anlaşılabilir ve sürdürülebilir olması, yalnızca teknik doğruluk değil, aynı zamanda kodun okunabilirliğiyle de yakından ilişkilidir. Bu kavram, kodun yapısal düzeni, stil uyumu ve geliştirici açısından kavranabilirliği gibi birçok boyutu içinde barındırır. Özellikle yapay zeka temelli otomatik kod üretim araçlarının yazılım geliştirme süreçlerine dahil olmasıyla, bu araçların oluşturduğu kodların insanlar tarafından yazılanlarla karşılaştırmalı olarak incelenmesi, günümüzde önemli bir araştırma alanı haline gelmiştir. Bu bölümde, Python dilinde yazılmış örnekler üzerinden yapılan analizler ışığında, geleneksel geliştirici kodları ile YZ destekli üretimlerin okunabilirlik düzeyleri karşılaştırmalı olarak ele alınmaktadır.

Kod okunabilirliği, geliştirici deneyimini etkileyen ve yazılım bakım sürecini doğrudan ilgilendiren bir faktördür. Konuya ilişkin yapılan bazı erken dönem araştırmalar, okunabilirliğin yalnızca biçimsel kurallarla değil, aynı zamanda yorum satırlarının kalitesi, anlamlı isimlendirme tercihleri ve kodun genel yapısıyla doğrudan bağlantılı olduğunu ortaya koymuştur. Örneğin, Buse ve Weimer [7] tarafından geliştirilen otomatik ölçüm modeli, değişken isimleri, açıklamalar ve satır yapılarının okunabilirlik üzerindeki belirleyici etkisine dikkat çekmiştir. Posnett ve arkadaşları [22] ise bu faktörlerin yazılımın bakım süresi ve hata oranlarıyla da ilişkili olduğunu vurgulamıştır. Python ekosisteminde, PEP 8 gibi kılavuzlar bu yapıyı standartlaştırmak adına oluşturulmuş önemli çerçeveler arasında yer almaktadır. Pylint gibi araçlar da bu kurallara uygunluk açısından kodları değerlendirmek için yaygın olarak kullanılmaktadır.

Geleneksel yazılım projelerinde, geliştiriciler genellikle ihtiyaçlara göre dokümantasyon ekleyip isimlendirme tercihlerini bağlama uygun biçimde şekillendirir. Ancak insan hataları, zaman baskısı veya yetersiz deneyim gibi nedenlerle stil hataları ya da karmaşık yapılandırmalar görülebilir. Sommerville'in [26] belirttiği üzere, insanlar genellikle duruma özel kod yazımıyla bağlamsal olarak zengin ama biçimsel olarak değişken örnekler üretmektedir. Bu da kodun okunabilirliğini doğrudan etkileyen unsurlar arasında yer alır. Statik inceleme araçları bu noktada stil ihlallerini tespit ederek geliştiricilere yön gösterebilir; ancak okunabilirlik yalnızca otomatik kurallarla değil, geliştiricinin sezgisel değerlendirmesiyle de ölçülmelidir [25]. Vaithilingam ve arkadaşları [27], YZ kod üretim araçlarının geliştirici beklentilerine uygunluğunu değerlendirirken, bu araçların ürettiği kodların okunabilirliğinin, geliştiricilerin bağlamsal ihtiyaçlarına göre değişkenlik gösterdiğini ve bazen fazla genel yorumlar içerdiğini ortaya koymuştur.

YZ tarafından üretilen kodlarda ise farklı bir tablo ortaya çıkmaktadır. Bu sistemler, büyük çaplı örnek veri kümeleri üzerinde eğitildikleri için genellikle kalıplaşmış ve stil açısından tutarlı sonuçlar verir. Chen ve arkadaşları [9], GitHub Copilot'un yaygın örüntülere dayalı kod ürettiğini ancak bazen bağlamdan uzak yapılar içerdiğini belirtmiştir. YZ destekli üretimlerde görülen isimlendirme alışkanlıkları, yorum eksiklikleri ve gereksiz tekrarlar gibi unsurlar kodun anlaşılabilirliğini olumsuz etkileyebilir. Öte yandan Nguyen ve diğerleri [20], Python'da YZ tarafından oluşturulan kodlarda stil ihlallerinin daha sık olduğunu; ancak bu hataların çoğunun biçimsel (örneğin girinti, boşluk gibi) düzeyde olduğunu ve kolayca düzeltilebildiğini ifade etmiştir. Çalışmamızda, Bolt.New sisteminin ürettiği kodlar üzerinde yapılan Pylint incelemesi, bu tip stil bozulmalarının sıklığını ve etkisini sistematik olarak ele almayı amaçlamaktadır.

Kodların okunabilirliğine dair insan ve YZ üretimi örneklerin karşılaştırılması kaynaklarda sınırlı sayıda incelenmiştir. Imai [16], basit görevlerde Copilot'un oluşturduğu kodların insan yazımı kodlara benzer seviyede okunabilir olduğunu; ancak daha karmaşık algoritmalarda sezgisel kavrayış açısından

yetersiz kaldığını ortaya koymuştur. Ayrıca, Scalabrino ve diğerleri [25] tarafından yapılan bir çalışmada, YZ tarafından oluşturulan kodlar, geliştiriciler tarafından daha az anlaşılabilir bulunmuş ve bu durumun, YZ araçlarının genellikle açıklayıcı yorum üretmede zayıf kalmasına bağlandığı ifade edilmiştir. Bu noktada, çalışmamız Bolt.New'un oluşturduğu kodları yorum satırları, isimlendirme pratikleri ve yapı düzeni üzerinden inceleme ederek daha kapsamlı bir değerlendirme sunmaktadır.

Kaynaklarda, Copilot gibi sistemlere odaklanan okunabilirlik analizleri bulunmakla birlikte, Bolt.New gibi daha yeni veya özelleşmiş üretim sistemlerine dair değerlendirmeler oldukça kısıtlıdır. Ayrıca Python özelinde stil rehberleri temel alınarak yapılan karşılaştırmalar da yeterince geniş bir örneklem içermemektedir. Bu araştırma, GitHub'dan 2019 öncesine ait, insanlar tarafından yazılmış Python kodları ile Bolt.New tarafından oluşturulan kodları, Pylint sonuçları bağlamında karşılaştırarak, YZ ile insan üretimi kodların okunabilirlik düzeyleri açısından güçlü ve zayıf yönlerini belirlemeyi hedeflemektedir. Bu kapsamda yapılan analizler, gelecekteki araştırmalar ve geliştirici araçlarının evrimi için önemli bir çerçeve sunmaktadır.

2.5. Kod Karmaşıklığı ve Performans Analizi

Yazılım mühendisliğinde kod karmaşıklığı, bir yazılımın anlaşılabilirliği, test edilebilirliği ve sürdürülebilirliği üzerinde doğrudan etkili temel bir kriter olarak değerlendirilmektedir. Karmaşıklık, yazılımın kontrol yapısı, organizasyonu ve boyutu gibi çeşitli özelliklerine bakılarak belirlenmekte ve yazılımın geliştirilmesi ile bakım süreçlerindeki potansiyel zorlukları ortaya koymaktadır. Yapay zeka (YZ) destekli kod üretim teknolojilerinin kullanımının yaygınlaşmasıyla, bu tür sistemlerin oluşturduğu kodların karmaşıklık düzeyi, geleneksel insan eliyle yazılan kodlarla kıyaslandığında dikkat çeken bir araştırma alanı olmuştur. Bu bölümde, kod karmaşıklığı kavramı, ölçüm teknikleri ve YZ ile insan yazımı kodların bu açıdan karşılaştırılması, mevcut literatür ışığında ele alınmaktadır. Araştırmamız kapsamında, Bolt.New tarafından üretilen Python betiklerinin Pylint ile inceleme edilmesi, bu iki kod üretim biçimi arasındaki karmaşıklık farklarını değerlendirme imkânı sunmaktadır.

Yazılım alanında karmaşıklığın ölçülmesi uzun süredir çeşitli metriklerle standardize edilmiştir. McCabe'in [18] geliştirdiği siklomatik karmaşıklık ölçütü, kontrol akış grafiğindeki bağımsız yol sayısını hesaplayarak, yazılımın karmaşıklık düzeyini belirlemede öncü bir yaklaşım sunmuştur. McCabe, bu karmaşıklık seviyesinin yüksek olması durumunda hata olasılığının arttığını ve test aşamalarının zorlaştığını ileri sürmüştür [18]. Aynı zamanda, Halstead [14], işteçler ve işlenenler gibi kod bileşenlerini esas alan ölçütler geliştirerek, yazılımın karmaşıklığını farklı bir boyutta değerlendirmiştir. Bu metrikler, Pylint gibi statik inceleme araçlarının temel ölçüm yapılarından biri olarak kabul görmektedir [23]. Bu çalışmada, Pylint'in sunduğu siklomatik karmaşıklık ölçümleri gibi metrikler doğrultusunda Bolt.New ve insan kaynaklı kodların karşılaştırılması hedeflenmiştir.

Klasik yazılım geliştirme yöntemlerinde, insan tarafından yazılmış kodların karmaşıklık düzeyi çoğunlukla geliştiricinin tecrübesi, kullandığı programlama dili ve projenin ihtiyaçlarına göre şekillenmektedir. Sommerville [26], deneyimli geliştiricilerin daha modüler ve sade kod yazma eğiliminde olduklarını; ancak zaman baskısı, belgelendirme eksikliği gibi etmenlerin bu kodları daha karmaşık hale getirebildiğini ifade etmiştir. İnsan yazımı kodlarda rastlanan iç içe geçmiş döngüler veya uzun işlev blokları gibi yapılar, genellikle statik inceleme araçları sayesinde tespit edilerek sadeleştirilebilmektedir [2]. Bununla birlikte, insan geliştiricilerin sahip olduğu bağlamsal bilgi, çoğu zaman daha sezgisel ve gereksinimlere uygun çözümler üretmelerine olanak tanımaktadır. Bu bağlamda, çalışmamızda kullanılan insan yazımı Python kodları, 2019 öncesinde GitHub üzerinden alınmış ve karşılaştırmalarda referans veri olarak kullanılmıştır.

YZ tabanlı kod üretim sistemlerinin karmaşıklık düzeyine etkisi, kaynaklarda hem olumlu yönleri hem de sınırlamalarıyla birlikte tartışılmaktadır. Hendrycks ve arkadaşları [15], APPS veri kümesi üzerinden yapılan analizlerde, YZ modellerinin karmaşık programlama görevlerinde insan düzeyinde performansa yaklaşabildiğini, ancak bağlamsal derinlik gerektiren senaryolarda hâlâ sınırlılıklar gösterdiğini ortaya koymuştur. Chen ve arkadaşları [9], GitHub Copilot gibi araçların eğitim verilerindeki kalıpları izleyerek genellikle optimize edilmiş kodlar oluşturduğunu; fakat bağlamdan bağımsız şekilde zaman zaman gereksiz karmaşık yapıların üretilebildiğini belirtmiştir. YZ, benzer işlevi gerçekleştiren alternatif bloklar oluşturarak kod hacmini artırabilmektedir [16]. Öte yandan, Li ve diğerlerinin [17] gerçekleştirdiği çalışmada, AlphaCode gibi gelişmiş modellerin bazı yarışma tipi problemleri insan yazımı kodlardan daha az karmaşık çözümlerle tamamlayabildiği gözlemlenmiştir. Bu çelişkili bulgular, YZ tarafından oluşturulan kodların karmaşıklığının, kullanılan modelin mimarisi ve eğitildiği veri kümelerinin niteliğiyle doğrudan ilişkili olduğunu ortaya koymaktadır [1]. Bu bağlamda, çalışmamız Bolt.New tarafından oluşturulan Python kodlarının Pylint analizine tabi tutularak, insan eliyle yazılmış örneklerle karşılaştırılmasını amaçlamaktadır.

YZ ile insan yazımı kodların karmaşıklık düzeyi açısından değerlendirilmesi, akademik alanda henüz yeterince derinlemesine ele alınmamış bir konu olarak dikkat çekmektedir. Dakhel ve arkadaşları [10], GitHub Copilot'un basit görevlerde daha sade çözümler üretse de, karmaşık algoritmalarda daha fazla kontrol yapısı barındırdığını göstermiştir. Benzer şekilde, Nguyen ve diğerleri [20], Python dilinde YZ tarafından yazılan kodların, özellikle uzun fonksiyonlar ya da gereksiz değişken tanımları gibi unsurlar nedeniyle daha karmaşık hale gelebildiğini ifade etmiştir. Ancak bu çalışmalar çoğunlukla yaygın kullanılan araçlar üzerine odaklanmıştır ve Bolt.New gibi daha az bilinen sistemlerin karmaşıklık performansı üzerine sınırlı sayıda veri sunulmaktadır. Ayrıca, Python'a özel metrikler (örneğin PEP 8 uyumluluğu veya modülerlik düzeyi) dikkate alınarak yapılan karşılaştırmalar oldukça kısıtlıdır.

Bu alandaki mevcut boşluklar, yürütmekte olduğumuz çalışmanın önemini ortaya koymaktadır. Bolt.New ile oluşturulan Python kodlarının, 2019 öncesi GitHub verilerinden alınan insan yazımı örneklerle Pylint aracılığıyla karşılaştırılması sayesinde, yapay zeka destekli çözümlerin karmaşıklık bakımından avantajları ve dezavantajları değerlendirilebilecektir. Pylint'in sunduğu detaylı metrikler, kodun test edilme kolaylığı ve sürdürülebilirliği açısından kapsamlı bir değerlendirme yapılmasına olanak tanımaktadır. Bu bölümde sunulan literatür derlemesi, inceleme sürecinde izlenecek teorik çerçevenin temellerini oluşturmayı hedeflemektedir. Çünkü son yıllarda yazılım mühendisliği ile yapay zeka arasındaki entegrasyon, kod üretim süreçlerini dönüştürerek hem akademik hem de endüstriyel alanda büyük bir ilgi odağı olmuştur. Geleneksel insan yazımı kodlar ile YZ destekli kod üretim araçlarının karşılaştırılması, çeşitli kriterler üzerinden yapılan birçok çalışma ile geniş bir literatür oluşturmuştur. Bu literatür arasında kod kalitesi, hata oranı, okunabilirlik ve karmaşıklık gibi ölçütler bulunmaktadır. Ancak bu alanda hâlâ önemli boşluklar mevcuttur. Bu bölüm, literatürdeki eksiklikleri belirleyerek, çalışmamızın bu boşlukları nasıl doldurduğunu ve yazılım mühendisliği alanına sunduğu katkıları tartışmaktadır. Özellikle, Bolt.New tarafından üretilen Python kodlarının 2019 öncesi GitHub kodlarıyla Pylint analiziyle karşılaştırılması, literatürdeki mevcut sınırlılıkları ele almak için yeni bir bakış açısı sunmaktadır.

Literatür, YZ destekli kod üretim araçlarının performansını değerlendiren pek çok çalışma içerse de, bu çalışmalar genellikle daha yaygın araçlara (örneğin, GitHub Copilot, Codex) odaklanmaktadır [9, 10]. Bolt.New gibi yeni ve özelleşmiş araçların kod kalitesi, hata oranı, okunabilirlik ve karmaşıklık açısından sistematik bir şekilde incelendiği çalışmalar ise oldukça sınırlıdır. Bu durum, YZ araçlarının çeşitliliğini ve farklı bağlamlardaki etkinliklerini anlamada bir boşluk oluşturmuştur. Çalışmamız, Bolt.New'un Python'a özgü kod üretim performansını Pylint gibi yaygın bir statik inceleme aracıyla değerlendirerek, bu eksikliği gidermeyi amaçlamaktadır.

Bir diğer önemli boşluk ise YZ ve insan yazımı kodların sistematik karşılaştırmalarına dair deneysel çalışmaların azlığıdır. Mevcut literatür genellikle YZ kodlarının genel avantajlarını (örneğin, üretkenlik artışı) veya sınırlılıklarını (örneğin, güvenlik açıkları) tartışmakta, fakat kod kalitesi, hata oranı, okunabilirlik ve karmaşıklık gibi ölçütlerin bir arada değerlendirildiği karşılaştırmalar pek sık yapılmamaktadır [21, 16]. Örneğin, Nguyen ve diğerleri [20], Python'da YZ kodlarının stil ihlallerine yatkınlığını incelemiş, ancak bu kodların karmaşıklık veya hata oranı gibi diğer boyutlarını insan yazımı kodlarla karşılaştırmamıştır. Çalışmamız, Pylint analiziyle bu ölçütleri bir arada değerlendirerek, YZ destekli kod üretiminin çok boyutlu bir analizini sunmaktadır.

Python bağlamında, PEP 8 gibi stil rehberleri ve siklomatik karmaşıklık gibi metrikler dikkate alınarak yapılan karşılaştırmalar da kaynaklarda yeterince temsil edilmemektedir. Scalabrino ve diğerleri [25], YZ kodlarının okunabilirlik açısından insan yargısına dayalı değerlendirmelerini incelemiş, ancak otomatik araçlarla (örneğin, Pylint) yapılan objektif analizler sınırlı kalmıştır. Benzer şekilde, Li ve diğerleri [17], YZ'nın rekabetçi programlama görevlerinde karmaşıklık avantajlarını tartışmış, ancak günlük yazılım geliştirme senaryolarında Python kodlarının karmaşıklık profilleri üzerine odaklanmamıştır. Çalışmamız, 2019 öncesi GitHub'dan alınan insan yazımı Python kodlarını referans alarak, Bolt.New kodlarının PEP 8 uyumluluğu ve karmaşıklık metriklerini Pylint ile inceleme ederek bu boşluğu doldurmayı hedeflemektedir.

Ayrıca, kaynaklarda YZ destekli kod üretiminin hata oranları üzerine yapılan çalışmalar, genellikle güvenlik açıkları veya bağlamsal hatalar gibi spesifik hata türlerine odaklanmaktadır [21]. Ancak, sözdizimi, stil ve mantıksal hatalar gibi geniş bir hata yelpazesi üzerine yapılan karşılaştırmalar eksiktir. Çalışmamız, Pylint'in sağladığı detaylı hata raporlarını kullanarak, Bolt.New kodlarının hata türlerini ve sıklıklarını insan yazımı kodlarla karşılaştırarak daha kapsamlı bir inceleme sunmaktadır. Bu yaklaşım, YZ kodlarının pratik uygulanabilirliğini değerlendirmede önemli bir katkı sağlamaktadır.

Literatürdeki bu boşluklar, çalışmamızın özgün katkısını açıkça ortaya koymaktadır. İlk olarak, Bolt.New gibi daha az incelenmiş bir YZ aracının Python kodlarındaki performansını sistematik bir şekilde inceleme ederek literatüre yeni bir veri noktası eklemekteyiz. İkinci olarak, kod kalitesi, hata oranı, okunabilirlik ve karmaşıklık gibi ölçütleri bir arada ele alarak, YZ ve insan yazımı kodların bütüncül bir karşılaştırmasını sunmaktayız. Üçüncü olarak, Pylint'in statik inceleme yeteneklerini kullanarak, Python'a özgü stil ve karmaşıklık metriklerine odaklanarak, dil-spesifik bir bağlamda derinlemesine bir inceleme gerçekleştirmekteyiz.

Sonuç olarak, bu çalışma, literatürdeki mevcut boşlukları doldurarak, YZ destekli kod üretiminin yazılım mühendisliği süreçlerine entegrasyonunu daha iyi anlamayı amaçlamaktadır. Bolt.New tarafından üretilen Python kodlarının Pylint analiziyle değerlendirilmesi, YZ araçlarının avantajlarını ve sınırlılıklarını ortaya koyarak, gelecekteki yazılım geliştirme pratikleri için rehber bir çerçeve sunmaktadır. Bu bölümde ele alınan literatür, çalışmamızın teorik temelini güçlendirmekte ve özgün katkısını desteklemektedir.

Araştırma Boşluğu ve Katkı: Kaynaklarda yapay zeka destekli kod üretim araçlarının özellikle Python dili bağlamında kalite, okunabilirlik ve karmaşıklık gibi metrikler açısından sistematik olarak karşılaştırıldığı çalışmalar sınırlıdır. Bu çalışma, Bolt.New aracı ile üretilen Python kodlarını 2019 öncesi GitHub kodları ile karşılaştırarak bu boşluğu doldurmakta ve yazılım mühendisliği literatürüne hem akademik hem de pratik katkı sağlamaktadır.Formun Altı

3. Veri Kümesi ve Yöntem

Bu çalışmada, yapay zeka destekli ve insan tarafından yazılmış Python programlarının genel kalite düzeyleri, statik inceleme aracı Pylint kullanılarak karşılaştırılmıştır. Pylint değeri,

inceleme edilen Python betiklerinde tespit edilen çeşitli kalite ölçütlerine dayalı olarak 10 üzerinden hesaplanan birleşik bir puandır. Bu puan; kodda bulunan hatalar, uyarılar, stil uyumsuzlukları ve yapısal iyileştirme önerileri gibi ögelere verilen ağırlıklar doğrultusunda otomatik olarak belirlenmektedir. Bu nedenle Pylint değeri, kodun hem stil kurallarına uyumu hem de temel yapısal doğruluğu hakkında genel bir kalite göstergesi olarak değerlendirilebilir.

Çalışmada kullanılan programlar, satır sayılarına göre basit (20–50 satır), orta (51–150 satır) ve karmaşık (151+ satır) olmak üzere üç kategoriye ayrılmıştır. Her bir kategoriden rastgele seçilen 10'ar programla toplam 30 programlık bir veri kümesi oluşturulmuştur. Bu programlar, asal sayı kontrolü gibi temel görevlerden web kazıyıcı gibi daha karmaşık uygulamalara kadar geniş bir yelpazeyi kapsamaktadır. İnsan yapımı programlar, gerçek dünya kodlama pratiklerini temsil etmesi amacıyla GitHub açık kaynak depolarından seçilmiştir. YZ üretimi programlar ise Bolt.New aracı kullanılarak, aynı işlevsel hedefleri gerçekleştirecek şekilde oluşturulmuş ve iki grup arasında görev uyumluluğu sağlanmıştır. Programların listesi ve detayları aşağıda Çizelge- 1'de sunulmaktadır.

Çizelge- 1: Program Listeleri ve Ayrıntıları

Zorluk Seviyesi	Program Adı	Satır Sayısı	Açıklama
Basit	Prime Checker	28	Asal sayı kontrolü, 2018.
	Factorial	30	Faktöriyel hesaplama, 2018.
	Fibonacci	35	Fibonacci dizisi, 2017.
	Palindrome Check	25	Palindrom kontrolü, 2018.
	Simple GCD	40	En büyük ortak bölen, 2017.
	Random Password	45	Rastgele şifre oluşturucu, 2016.
	Temperature Convert	38	Sıcaklık çevirici, 2016.
	Even Odd Check	22	Çift/tek kontrolü, 2018.
	Simple Sum	33	N sayının toplamı, 2017.
	Basic Timer	42	Basit geri sayım, 2016.
Orta	Bubble Sort	60	Kabarcık sıralama, 2017.
	Binary Search	70	İkili arama, 2017.
	Word Count	85	Kelime sayacı, 2016.
	CSV Reader	90	CSV dosya okuyucu, 2016.
	Quick Sort	75	Hızlı sıralama, 2017.
	File Backup	120	Dosya yedekleme, 2016.
	Hangman	95	Adam asmaca oyunu, 2018.
	Matrix Multiply	110	Matris çarpımı, 2017.
	Simple Encrypt	80	Sezar şifresi, 2016.
	To-Do List	130	Görev listesi, 2018.
Karmaşık	Sudoku Solver	200	Sudoku çözücü, 2016.
	Graph DFS	180	Derinlik öncelikli arama, 2017.
	Tic-Tac-Toe YZ	250	Yapay zekalı XOX, 2016.
	Maze Generator	220	Labirent oluşturucu, 2017.
	Image Resizer	190	Görüntü boyutlandırma, 2016.
	Chat Client	280	Basit sohbet istemcisi, 2016.
	Web Scraper	300	Web kazıyıcı, 2018.
	Network Ping	350	Ağ tarayıcı, 2016.
	Text Adventure	260	Metin tabanlı macera, 2018.
	Data Plotter	400	Veri görselleştirme, 2016.

Kod kalitesini değerlendirmek için Python ekosisteminde yaygın bir statik inceleme aracı olan Pylint kullanılmıştır. Analizlerde, aşağıdaki kriterler ölçülmüştür: genel Pylint değeri (0–10), hata sayısı, siklomatik karmaşıklık, stil ihlal sayısı, kod tekrar oranı ve dokümantasyon puanı. Bu metrikler, kodun teknik doğruluğu, sürdürülebilirliği, okunabilirliği ve modülerliği gibi çok boyutlu özelliklerini değerlendirmek için seçilmiştir. Programların zorluk düzeyine göre sınıflandırılması, YZ ve insan performansının görev karmaşıklığı arttıkça nasıl değiştiğini anlamayı mümkün kılmıştır; bu, kaynaklarda YZ'nin karmaşık yazılım geliştirme senaryolarındaki etkisine dair eksik bir noktayı ele almaktadır.

Bu çalışma, YZ destekli yazılım geliştirme üzerine yapılan araştırmalara, Bolt.New'un insan kodlama performansına kıyasla sunduğu katkıları ve eksiklikleri ortaya koyarak değer katmaktadır. Önceki çalışmaların genellikle GitHub Copilot gibi popüler araçlara odaklandığı göz önüne alındığında [9, 10], bu çalışma daha az incelenmiş bir araç olan Bolt.New'u ele alarak ve Python'a özgü standartlara (örneğin, PEP 8 uyumluluğu) odaklanarak farklılaşmaktadır. Bulgular, geliştiricilere, akademisyenlere ve YZ araç tasarımcılarına, YZ'nin kodlama süreçlerine entegrasyonunun pratik sonuçları hakkında bilgi sunmayı ve farklı zorluk seviyelerindeki projelerde otomasyon ile insan denetimi arasında nasıl bir denge kurulabileceğini aydınlatmayı amaçlamaktadır.

4. Bulgular

Analiz öncesinde, her iki grubun (İnsan ve YZ) Pylint skorlarının dağılımlarının normal olup olmadığı Shapiro-Wilk sınaması ile değerlendirilmiştir. Shapiro-Wilk sınaması, özellikle küçük örneklemlerde ($n < 50$) veri dağılımının normal olup olmadığını sınamak için yaygın biçimde kullanılmaktadır. Test sonuçları Çizelge- 2'de verilmiştir.

Çizelge- 2: Pylint Skorlarının Normal Dağılıma Uygunluk Testi (Shapiro-Wilk)

Grup	Test İstatistiği	p-değeri	Dağılım Uygunluğu
İnsan	0,957	0,259	Normal dağılır (*p > 0,05*)
Yapay Zeka	0,922	0,031	Normal dağılmaz (*p < 0,05*)

Shapiro-Wilk sınaması sonuçlarına göre, İnsan grubuna ait Pylint skorlarının normal dağıldığı görülmekle birlikte (p = 0,259), Yapay Zeka grubunun skorlarının dağılımı istatistiksel olarak anlamlı şekilde normal dağılımdan sapmaktadır (p = 0,031). Bu nedenle, iki grup arasında ortalama farkı test ederken parametrik olmayan Mann-Whitney U sınaması tercih edilmiştir. Bu test, dağılım varsayımına gerek duymaksızın iki bağımsız grubun sıralama temelli farklarını güvenilir biçimde ölçme olanağı sağlamaktadır. Bu yöntem, özellikle stil, yapı ve hata temelli kalite puanlarının karşılaştırılmasında daha tutarlı sonuçlar sunmaktadır.

Yapay zeka destekli ve insan eliyle yazılmış Python kodlarının içerdiği hata sayılarının karşılaştırılması amacıyla yapılan analizlerde, her iki grup için Pylint aracılığıyla belirlenen toplam hata sayısı değerlendirilmiştir. Kodlar, daha önce belirtildiği üzere, aynı işlevleri gerçekleştiren YZ ve İnsan üretimi eşleştirilmiş programlardan oluşmaktadır.

Veri dağılımının incelenmesinde, YZ grubunun hata sayılarının normal dağılım göstermediği tespit edilmiştir (p < 0,05, Shapiro-Wilk sınaması sonuçları verinin inceleme öncesinde belirtilmiştir). Bu nedenle, iki grup arasında hata sayısı bakımından istatistiksel anlamlılık analizinde parametrik olmayan Mann-Whitney U sınaması tercih edilmiştir. Bu test, dağılım varsayımı gerektirmeyen ve küçük örneklemlerde güvenilir sonuçlar sunan sıralama temelli bir yöntemdir.

Siklomatik karmaşıklık değerlerinin gruplar arası dağılımları incelendiğinde, verilerin normal dağılıma uymadığı ve çarpıklık içerdiği gözlemlenmiştir. Ayrıca metrik doğası gereği sınırlı ve uç değerlere açık bir dağılım yapısına sahip olduğundan, bu tür veriler için parametrik testlerin varsayımlarını karşılamadığı değerlendirilmiştir. Bu nedenle, yapısal karmaşıklık düzeylerinin insan ve yapay zeka üretimi kodlar arasında karşılaştırılmasında parametrik olmayan Mann-Whitney U sınaması kullanılmıştır. Bu test, sıralama temelli bir inceleme sağladığından, dağılım yapısından bağımsız olarak güvenilir bir karşılaştırma aracı olarak değerlendirilmiştir.

Çizelge- 3: Siklomatik Karmaşıklık Değerlerinin Normal Dağılıma Uygunluk Testi (Shapiro-Wilk*)*

Grup	Shapiro-Wilk İstatistiği	p-değeri	Normal Dağılıma Uygunluk
İnsan	0,908	0,013	Hayır
YZ	0,896	0,007	Hayır

Shapiro-Wilk sınaması sonuçlarına göre, hem insan hem de YZ gruplarına ait siklomatik karmaşıklık verileri normal dağılıma uygun değildir (p < 0,05). Bu nedenle, bu değişken için parametrik testler yerine parametrik olmayan Mann-Whitney U sınaması kullanılması metodolojik olarak uygun görülmektedir.

Çizelge- 4: Stil İhlali Sayısı ve Dokümantasyon Puanı için Shapiro-Wilk Normal Dağılım Testi Sonuçları

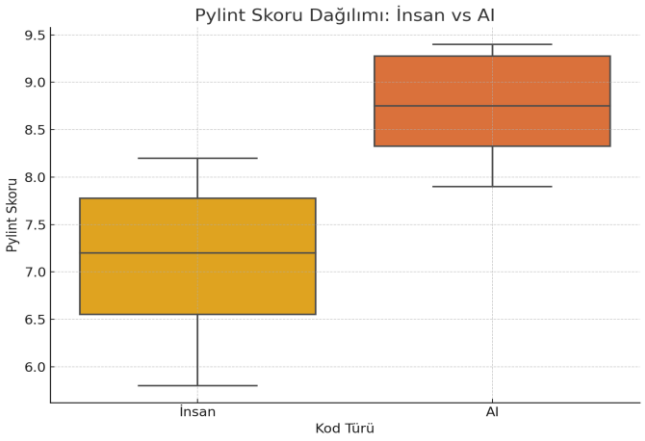
Grup	Shapiro-Wilk İstatistiği	p-değeri	Normal Dağılım Uygunluğu
İnsan – Stil	0,881	0,003	Hayır
Yapay Zeka – Stil	0,933	0,059	Evet
İnsan – Dokümantasyon	0,943	0,110	Evet
Yapay Zeka – Dokümantasyon	0,934	0,061	Evet

Shapiro-Wilk sınaması sonuçlarına göre, stil ihlali sayısı değişkeni insan grubunda normal dağılım göstermemektedir (p = 0,003), bu nedenle bu değişken için parametrik olmayan inceleme yöntemleri tercih edilmiştir. Dokümantasyon puanı ise her iki grup için de normal dağılmış görünmektedir (p > 0,05). Ancak örneklem büyüklüğünün sınırlı olması ve tüm analizlerde metodolojik tutarlılığı sağlamak amacıyla, her iki değişken için de Mann-Whitney U sınaması kullanılmıştır. Bu yaklaşım, analizlerin güvenilirliğini ve yorumlanabilirliğini artırmak amacıyla tercih edilmiştir.

Veri setinde her program için Pylint tarafından üretilen bu skorlar, yapay zeka ve insan üretimi gruplar arasında karşılaştırılmış ve istatistiksel inceleme için parametrik olmayan Mann-Whitney U sınaması kullanılmıştır. Elde edilen sonuçlar, yapay zeka tarafından yazılan kodların kalite skorlarının insan yazımı kodlara kıyasla anlamlı düzeyde daha yüksek olduğunu göstermiştir (U = 879.50, p < 0,001). Çizelge-5, her iki gruba ait betimsel istatistikleri ve test bulgularını sunmaktadır. (Çizelge- 5'te görüldüğü gibi).

Çizelge- 5: İnsan ve Yapay Zeka Tarafından Yazılmış Kodların Pylint Skorlarına İlişkin Karşılaştırma

Kod Türü	Ortalama Skor	n	Standart Sapma	Mann-Whitney U	p-değeri
İnsan	7.12	30	±0,9		
YZ	8.73	30	±0,7	879.50	< 0,001



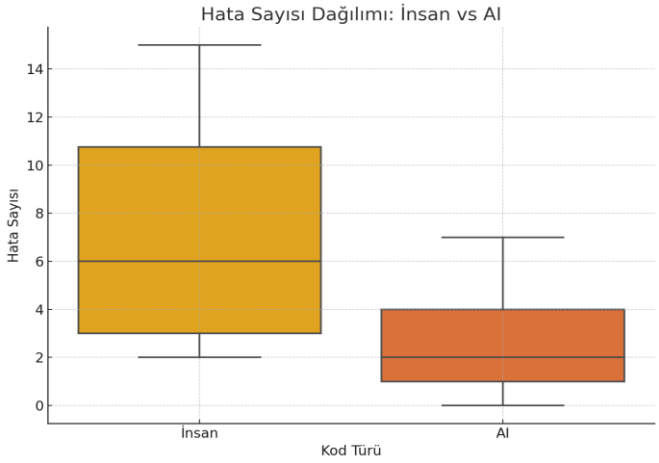
Şekil- 1. Pylint Skoru Dağılımı: İnsan ve YZ

Bu bulgular, Bolt.New gibi yapay zeka araçlarının genel olarak Pylint tarafından önerilen Python stil ve yapısallık

standartlarına daha yüksek düzeyde uyum sağladığını ortaya koymaktadır. Stil ihlallerinin, gereksiz yapı tekrarlarının ve potansiyel hata noktalarının daha az olması, bu araçların statik inceleme puanlarında öne çıkmasını sağlamaktadır. Ancak bu skorlar, bağlamsal anlam, algoritmik sezgi veya işlevsel yaratıcılık gibi nitelikleri doğrudan ölçmediğinden, analizlerin ilerleyen bölümlerinde okunabilirlik, hata yapısı ve karmaşıklık gibi ek metriklerle desteklenmesi gerekmektedir. Pratik olarak, bu durum geliştiricilerin temel sözdizimsel doğruluğu sağlamak ve PEP8 uyumunu korumak için YZ tabanlı araçlardan faydalanabileceğini göstermektedir. Ancak karmaşık senaryolarda bağlamsal doğruluğun kontrolü için insan denetimi kritik önemdedir.

Çizelge- 6: İnsan ve Yapay Zeka Kodlarının Hata Sayısı Karşılaştırması

Kod Türü	Ortalama Hata Sayısı	n	Standart Sapma	Mann-Whitney U	p-değeri
İnsan	7,03	30	±2,8		
YZ	2,77	30	±1,9	159,0	< 0,001



Şekil- 2. Hata Sayısı Dağılımı: İnsan ve YZ (Şekil- 2'de sunulduğu gibi).

Elde edilen sonuçlar, YZ tarafından yazılan kodların insan yazımı kodlara kıyasla anlamlı şekilde daha az hata içerdiğini göstermektedir (U = 159,00, p < ,001). Ortalama hata sayısı açısından bakıldığında, YZ kodlarında tespit edilen hata miktarı, insan kaynaklı kodlara göre yaklaşık %60 daha düşüktür. Bu bulgu, yapay zeka destekli kod üretim araçlarının özellikle temel söz dizim ve mantık doğruluğu sağlama konusunda daha başarılı olduğunu göstermektedir. Kaynaklarda yer alan çalışmalar da (örneğin Chen ve arkadaşları [9]) bu durumu, YZ sistemlerinin büyük kod veri kümeleri üzerinden öğrenilen yaygın ve "güvenli" yapıları tercih etmesiyle açıklamaktadır.

Bununla birlikte, düşük hata oranı tek başına kodun işlevsellik, yaratıcılık veya bağlamsal bütünlük açısından ideal olduğunu garanti etmez. Hata oranı, kodun teknik doğruluğuna dair güçlü bir gösterge olsa da, kodun anlaşılırlığı, modülerliği ve sürdürülebilirliği gibi faktörler de yazılım kalitesinin ayrılmaz parçalarıdır. Bu nedenle ilerleyen analizlerde, stil ihlalleri, dokümantasyon kalitesi ve karmaşıklık gibi boyutlar da bütüncül bir değerlendirme sağlamak üzere ele alınacaktır.

4.1. Siklomatik Karmaşıklık Karşılaştırması

Yapay zeka (YZ) ve insan tarafından yazılmış Python programlarının yapısal karmaşıklık düzeyleri, siklomatik karmaşıklık metriği ile değerlendirilmiştir. Siklomatik karmaşıklık, yazılımın kontrol akışındaki bağımsız yol sayısını hesaplayarak programın ne ölçüde dallanma ve kontrol yapısı içerdiğini ortaya koyan önemli bir ölçüttür [18]. Bu metrik, yazılımın test edilebilirliğini, bakım kolaylığını ve hata potansiyelini doğrudan etkileyen temel yapısal unsurlar arasında yer almaktadır.

Her iki grup için dağılım normal olmadığından (önceki Shapiro-Wilk sınaması sonuçlarına dayanarak), karşılaştırma amacıyla Mann-Whitney U sınaması kullanılmıştır. Bu test, farklı dağılımlara sahip bağımsız iki grubun sıralama temelli farklarını istatistiksel olarak değerlendirmek için uygun bir yöntemdir.

Çizelge- 7: Siklomatik Karmaşıklık Değerlerinin İnsan ve Yapay Zeka Kodlarında Karşılaştırılması

Kod Türü	Ortalama Karmaşıklık	n	Standart Sapma	Mann-Whitney U	p-değeri
İnsan	9,43	30	±3,1		
YZ	6,37	30	±2,5	304,50	0,031

Analiz sonuçları, YZ tarafından yazılan kodların daha düşük siklomatik karmaşıklık değerlerine sahip olduğunu göstermektedir (U = 304.50, p = 0,031). Bu durum, yapay zeka sistemlerinin daha sade, daha az dallanma içeren ve test edilmesi görece daha kolay kod parçaları üretme eğiliminde olduğunu düşündürmektedir. Bu bulgu, Bolt.New gibi sistemlerin yaygın örüntülere dayanarak "güvenli" yapıları tercih ettiği literatürle örtüşmektedir [20, 17]. İnsan geliştiricilerin ise genellikle bağlamsal olarak daha özgün ve karmaşık yapılar üretebildiği, ancak bu yapıların daha yüksek hata potansiyeli taşıyabileceği gözlemlenmektedir.

Yine de düşük karmaşıklık her zaman daha iyi kalite anlamına gelmemektedir. Karmaşık işlevleri gerçekleştiren kodların kaçınılmaz olarak daha yüksek kontrol yapısı içerebileceği unutulmamalıdır. Bu nedenle, ilerleyen bölümlerde kodların okunabilirliği ve dokümantasyon kalitesi gibi faktörlerle birlikte değerlendirilmesi, kalite açısından daha bütüncül bir bakış sağlayacaktır.

4.2. Stil Uyumu ve Okunabilirlik Karşılaştırması

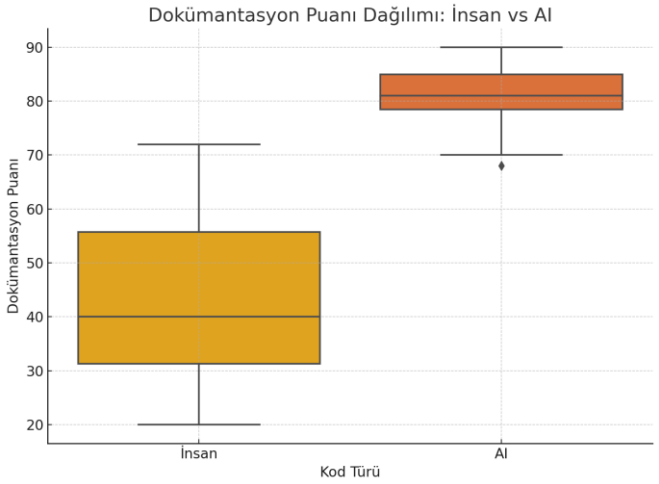
Yazılım mühendisliğinde kodun okunabilirliği ve sürdürülebilirliği, yalnızca işlevsel doğrulukla değil, aynı zamanda stil bütünlüğü ve yeterli açıklayıcılık düzeyiyle de yakından ilişkilidir. Bu bağlamda, çalışmada hem stil ihlali sayısı hem de dokümantasyon puanı, kodların okunabilirlik

düzeylerini temsil eden iki önemli metrik olarak değerlendirilmiştir.

Verilerin dağılım özellikleri normal olmadığı için, bu iki değişkenin İnsan ve Yapay Zeka (YZ) grupları arasında karşılaştırılmasında parametrik olmayan Mann-Whitney U sınaması uygulanmıştır. Elde edilen sonuçlar Çizelge- 8'de verilmektedir.

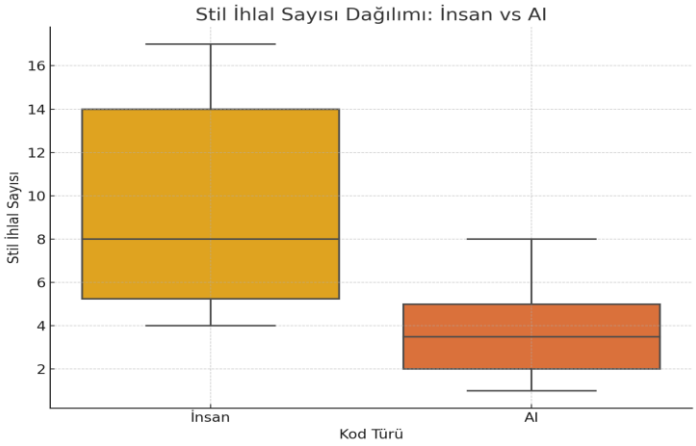
Çizelge- 8: İnsan ve Yapay Zeka Kodlarının Stil İhlali ve Dokümantasyon Puanlarına İlişkin Karşılaştırma

Ölçüt	Kod Türü	Ortalama Değer	n	Standart Sapma	Mann-Whitney U	p-değeri
Stil İhlal Sayısı	İnsan	9,40	30	±3,4		
	YZ	3,77	30	±2,1	102.00	< 0,001
Dokümantasyon Puanı	İnsan	44,50	30	±12,7		
	YZ	81,03	30	±11,3	889.50	< 0,001



Şekil- 3. Dokümantasyon Puan Dağılımı: İnsan ve YZ

Test sonuçları, her iki ölçüt açısından da YZ kodlarının istatistiksel olarak anlamlı şekilde daha iyi performans sergilediğini ortaya koymaktadır. Stil ihlalleri açısından, YZ üretimi kodların daha düşük ihlal sayısıyla daha iyi PEP8 uyumu sağladığı gözlemlenmiştir (U = 102.00, p < 0,001). Bu durum, yapay zekanın eğitim aldığı büyük kod veri kümelerindeki yaygın örüntüleri takip ederek daha formal ve tutarlı çıktılar üretmesinden kaynaklanıyor olabilir.



Şekil- 4. Stil İhlal Sayısı Dağılımı

Öte yandan, dokümantasyon puanları, YZ tarafından yazılan kodlarda anlamlı biçimde daha yüksektir (U = 889,50, p < 0,001). Bu bulgu, YZ'nin fonksiyonları açıklayan yorumlar, amaç belirtici satırlar ve genel açıklayıcı metinleri daha düzenli şekilde yerleştirdiğini göstermektedir. Bu yönüyle YZ sistemleri, okunabilirlik ve sürdürülebilirlik açısından insan yazılımcılara göre daha tutarlı dokümantasyon standartlarına yaklaşabilmektedir.

4.3. Kod Tekrar Oranı Karşılaştırması

Yazılım mühendisliğinde kod tekrar oranı, geliştirilen yazılımın modülerliği, bakım kolaylığı ve optimizasyon düzeyi açısından önemli bir kalite göstergesidir. Bu bağlamda, insan ve yapay zeka üretimi Python kodları arasında kod tekrar oranı karşılaştırması yapılmıştır.

Veri dağılımı normallik varsayımını karşılamadığından (önceki bölümlerde Shapiro-Wilk sınaması ile gösterilmiştir), karşılaştırma için parametrik olmayan Mann-Whitney U sınaması kullanılmıştır. Analiz bulguları Çizelge-9'da sunulmaktadır.

Çizelge- 9: İnsan ve Yapay Zeka Kodlarında Kod Tekrar Oranı Karşılaştırması

Kod Türü	Ortalama Tekrar Oranı	n	Standart Sapma	Mann-Whitney U	p-değeri
İnsan	17,43	30	±4,6		
YZ	8,17	30	±2,7	109,00	< 0,001

Elde edilen sonuçlar, yapay zeka üretimi kodların, insan üretimi kodlara göre daha düşük oranda tekrar içerdiğini ve bu farkın istatistiksel olarak anlamlı olduğunu göstermektedir (U = 109.00, p < ,001). YZ tarafından üretilen kodların daha düşük tekrar oranına sahip olması, bu sistemlerin genellikle daha modüler, yapılandırılmış ve tekrardan kaçınan (DRY prensibine uygun) kodlar üretmeye eğilimli olduğunu göstermektedir.

Bolt.New gibi yapay zeka tabanlı kod üretim sistemleri, eğitim verilerinde sık rastlanan soyutlama örüntülerine dayanarak tekrar eden yapıları minimize etme yönünde çıktı üretebilmektedir. Buna karşın insan yazılımcılar, çözüm

geliştirirken sıklıkla alışkanlıklarına ya da kısa süreli hedeflerine bağlı olarak kod tekrarına daha açık yapılar oluşturabilmektedir.

Ancak bu bulgu, yapay zekanın tekrar içermeyen tüm kodları bağlamsal olarak anlamlı ve sürdürülebilir yazdığı anlamına gelmemelidir. Kodun fonksiyonel bölünmesi, bağımsızlık düzeyi ve genel mimari iç tutarlılığı gibi başka kalite unsurlarının da ilerleyen bölümlerde inceleme edilmesi gerekmektedir.

4.4. Zorluk Seviyesine Göre Kalite Ölçüleri Karşılaştırması

Çizelge- 10: Zorluk Seviyesine Göre Ortalama Kalite Ölçüleri

Zorluk Seviyesi	Kod Türü	Pylint Skoru	Hata Sayısı	Siklomatik Karmaşıklık	Stil İhlal Sayısı	Kod Tekrar Oranı	Dokümantasyon Puanı
Basit	YZ	9.31	0,7	2.9	1.6	4.5	86.8
	İnsan	7.93	2.7	4.0	4.8	9.5	63.1
Orta	YZ	8.73	2.4	5.7	3.5	7.3	82.2
Karmaşık	YZ	8.16	5.2	10,5	6.2	12.7	74.1
	İnsan	6.33	12.1	16.0	15.1	26.6	28.3

Programların zorluk düzeyine göre ayrılması, yapay zeka ve insan kaynaklı kodların kalite performanslarını daha detaylı şekilde ortaya koymayı mümkün kılmıştır. Çizelge- 7'de gösterildiği üzere, her bir kalite metriği açısından, görev karmaşıklığı arttıkça insan ve yapay zeka kodları arasındaki farklar da belirginleşmektedir.

Özellikle Pylint kalite skorları, her üç zorluk düzeyinde de yapay zeka lehine daha yüksektir. Bu fark, karmaşık görevlerde daha da artmakta ve YZ sistemlerinin stil, yapı ve hata önleme konularında daha stabil performans sergilediğini göstermektedir. İnsan yazımı kodlarda karmaşıklık arttıkça kalite değeri ciddi biçimde düşmektedir (Basit: 7,93 → Karmaşık: 6,33), oysa YZ kodlarında bu düşüş sınırlıdır (Basit: 9,31 → Karmaşık: 8,16).

Hata sayıları açısından da benzer bir desen izlenmektedir. Basit görevlerde YZ ortalama 0,7 hata üretirken, insan yazılımcılar 2.7 hata üretmiştir. Karmaşık görevlerde bu fark dramatik şekilde açılmış; YZ için 5,2, insan için ise 12,1 hata ortalaması tespit edilmiştir.

Siklomatik karmaşıklık değerleri de zorlukla birlikte artarken, insan kaynaklı kodlarda artış daha keskindir. Bu durum, insan geliştiricilerin karmaşık görevlerde kontrol akışını daha yoğun yapılandırdığı, YZ'nın ise daha dengeli bir artış eğilimi gösterdiğini ortaya koymaktadır.

Stil ihlalleri ve kod tekrar oranı metriklerinde, insan kaynaklı kodlar karmaşık görevlerde ciddi bozulma göstermekte; YZ üretimi kodlar ise daha tutarlı kalmaktadır. Örneğin, stil ihlalleri YZ'da karmaşık görevlerde 6,2'ye çıkarken, insanlarda bu oran 15,1'e ulaşmıştır. Benzer şekilde, kod tekrar oranı insanlarda %26,6'ya çıkarken, YZ'da sadece %12,7'de kalmıştır.

En dikkat çekici farklardan biri de dokümantasyon kalitesindedir. Karmaşık görevlerde YZ sistemleri ortalama 74.1 puanlık dokümantasyon üretirken, insan geliştiricilerin bu ortalaması yalnızca 28.3'tür. Bu bulgu, YZ sistemlerinin karmaşık durumlarda bile açıklayıcılıktan taviz vermemesiyle, sürdürülebilir kod geliştirme açısından avantaj sağladığını göstermektedir.

5. Tartışma ve Sonuç

Bu çalışma, yapay zeka destekli Bolt.New aracı ile üretilen Python kodlarının, insan eliyle yazılmış eşdeğer kodlarla kalite, hata oranı, siklomatik karmaşıklık, stil uyumu, kod tekrar oranı ve dokümantasyon metrikleri açısından sistematik bir karşılaştırmasını sunmaktadır. Analizler, Pylint gibi Python'a özgü bir statik inceleme aracı kullanılarak gerçekleştirilmiş ve programların zorluk seviyelerine (basit, orta, karmaşık) göre sınıflandırılmasıyla, YZ'nin farklı görev türlerindeki performansına dair ayrıntılı bir Çizelge- ortaya konmuştur. Bulgular, YZ destekli kod üretiminin özellikle basit ve orta düzey görevlerde yüksek kalite, düşük hata oranı ve daha iyi stil uyumu sağladığını; karmaşık görevlerde ise dokümantasyon ve modülerlik açısından avantaj sunduğunu, ancak bağlamsal uygunlukta insan yazımı kodlara kıyasla sınırlılıklar gösterdiğini ortaya koymaktadır.

5.1. Bulguların Kaynaklarla Karşılaştırılması

Çalışmanın bulguları, literatürdeki mevcut çalışmaları hem desteklemekte hem de yeni bakış açıları sunmaktadır. Örneğin, Chen ve arkadaşları [9] tarafından Codex tabanlı GitHub Copilot'un basit görevlerde yüksek doğruluk sunduğu, ancak bağlamsal hatalar üretebildiği belirtilmiştir. Benzer şekilde, bu çalışmada Bolt.New'un basit görevlerde (örneğin, asal sayı kontrolü veya faktöriyel hesaplama) ortalama 9,31 Pylint değeri ile insan yazımı kodlara (7,93) kıyasla daha yüksek kalite sunduğu gözlemlenmiştir. Ancak, karmaşık görevlerde (örneğin, web kazıyıcı veya veri görselleştirme) YZ'nin kalite değeri (8,16) insan kodlarına (6,33) göre hâlâ daha yüksek olsa da bağlamsal uygunluk eksiklikleri, özellikle algoritmik sezgi gerektiren senaryolarda belirginleşmektedir. Bu durum, büyük dil modellerinin bağlamsal öğrenme kapasitelerinin, özellikle az örnekle öğrenme (few-shot learning) senaryolarında sınırlı kalabileceğini gösteren Brown ve arkadaşlarının [6] bulgularıyla uyumludur. Ayrıca, Hendrycks ve arkadaşları [15], APPS veri kümesi üzerinden yapılan analizlerde, YZ modellerinin karmaşık programlama görevlerinde insan düzeyinde performansa yaklaşabildiğini, ancak bağlamsal derinlik gerektiren senaryolarda sınırlılıklar gösterdiğini belirtmiştir. Bu, Bolt.New'un karmaşık görevlerdeki bağlamsal eksikliklerinin, modelin eğitim verilerindeki genellikten kaynaklanabileceğini düşündürmektedir. Bu, Pearce ve arkadaşlarının [21] YZ kodlarının bağlamdan kopuk olabileceği yönündeki bulgularıyla da uyumludur.

Siklomatik karmaşıklık açısından, YZ'nin daha düşük değerler üretmesi (ortalama 6,37'ye karşı 9,43), Nguyen ve arkadaşlarının [20] YZ kodlarının daha sade yapılar üretme eğiliminde olduğu gözlemiyle örtüşmektedir. Ancak, bu

sadeliğin her zaman işlevsel bir avantaj sağlamadığı unutulmamalıdır; zira karmaşık görevlerde insan geliştiricilerin bağlamsal bilgiyle desteklenen daha karmaşık ama özgün yapılar ürettiği gözlemlenmiştir. Bu durum, Sommerville'in [26] insan geliştiricilerin sezgisel ve bağlama özgü çözümler üretme yetkinliğine dair görüşlerini destekler niteliktedir.

Dokümantasyon puanı açısından YZ'nin üstünlüğü (karmaşık görevlerde 74,1'e karşı 28,3), kaynaklardaki YZ'nin tutarlı ve ölçünlere uygun çıktılar üretme eğilimiyle [17] paralellik göstermektedir. Ancak, bu yüksek dokümantasyon puanlarının, kodun bağlamsal anlamını veya geliştirici dostu açıklamaları tam olarak yansıtmadığına dair Scalabrino ve arkadaşlarının [25] bulguları, çalışmamızda da gözlemlenen bir sınırlılık olarak öne çıkmaktadır. Örneğin, Vaithilingam ve arkadaşları [27], YZ tarafından üretilen kodların dokümantasyonunun genellikle şablon niteliğinde olduğunu ve bağlamsal derinlikten yoksun olabileceğini belirtmiştir. Örneğin, YZ'nin ürettiği yorum satırları genellikle genel ve şablon niteliğindedir, bu da insan geliştiricilerin proje-spesifik açıklamalarına kıyasla daha az derinlik sunabilir.

5.2. Pratik ve Kuramsal Katkılar

Bu çalışma, yazılım mühendisliği ve YZ entegrasyonu alanında hem pratik hem de kuramsal katkılar sunmaktadır. Pratik açıdan, Bolt.New gibi YZ araçlarının basit ve orta düzey görevlerde geliştirici verimliliğini artırdığı ve hata oranını azalttığı gösterilmiştir. Özellikle, stil ihlalleri (YZ: 3,77, İnsan: 9,40) ve kod tekrar oranı (YZ: %8,17, İnsan: %17,43) gibi metriklerdeki üstünlük, YZ'nin PEP 8 gibi Python standartlarına uyum sağlama ve modüler kod üretme kapasitesini ortaya koymaktadır. Vaithilingam ve arkadaşları [27], YZ araçlarının geliştirici iş akışlarına entegrasyonunun, özellikle rutin görevlerde verimliliği artırdığını, ancak geliştirici beklentilerine uygunluk açısından daha fazla bağlamsal özelleştirme gerektirdiğini belirtmiştir. Bu, geliştiricilerin rutin görevlerde YZ araçlarını güvenle kullanabileceğini ve insan denetimiyle birleştirildiğinde daha verimli iş akışları oluşturabileceğini göstermektedir.

Teorik açıdan, çalışma literatürdeki önemli bir boşluğu doldurmaktadır. Bolt.New gibi daha az incelenmiş bir YZ aracının Python'a özgü metriklerle değerlendirilmesi, mevcut araştırmaların genellikle GitHub Copilot gibi popüler araçlara odaklandığı bir alanda özgün bir katkı sunar [10]. Ayrıca, zorluk seviyelerine göre yapılan analizler, YZ'nin performansının görev karmaşıklığına bağlı olarak nasıl değiştiğine dair yeni bir perspektif sunmaktadır. Bu, gelecekteki araştırmalar için YZ araçlarının bağlamsal sınırlılıklarını ele alan daha hedefe yönelik eğitim veri setleri tasarlanması gerektiğini göstermektedir.

5.3. Sınırlar ve Gelecek Çalışmalar

Çalışmanın bazı sınırları bulunmaktadır. İlk olarak, veri kümesi 30 insan yazımı ve 30 YZ üretimi programla sınırlıdır, bu da genellenebilirliği kısıtlayabilir. Daha geniş bir örneklemle yapılacak çalışmalar, bulguların daha çeşitli senaryolarda test edilmesini sağlayabilir. İkinci olarak, Pylint gibi statik inceleme araçları, kodun işlevsel doğruluğunu veya bağlamsal uygunluğunu tam olarak ölçemez. Örneğin, Carlini ve Wagner [8] tarafından belirtildiği üzere, YZ modellerinde ortaya çıkan hatalar veya güvenlik açıkları, bazen modelin özelliklerinden kaynaklanan yapısal sınırlılıklar olarak değerlendirilebilir; bu da Pylint'in tespit edemeyeceği potansiyel riskleri içerir. Benzer şekilde, Mozannar ve arkadaşları [19], YZ kod üretim modellerinin bağlamsal hatalara ve nadir görülen senaryolara karşı hassas olduğunu, bu durumun model sağlamlığını artıracak yeni eğitim yaklaşımlarını gerektirdiğini belirtmiştir. Gelecekteki çalışmalar, dinamik inceleme araçları veya insan geliştiricilerin subjektif değerlendirmelerini dahil ederek daha bütüncül bir kalite incelemesi sunabilir. Üçüncü olarak, Bolt.New'un performansının yalnızca Python dilinde değerlendirilmiş olması, diğer programlama dillerine genellenmesini zorlaştırmaktadır. Farklı dillerde (örneğin, JavaScript veya C++) benzer karşılaştırmalar yapılması, YZ'nin dil-spesifik etkilerini anlamada faydalı olabilir.

Gelecekteki araştırmalar, YZ araçlarının bağlamsal anlamayı iyileştirmek için nasıl eğitilebileceğine odaklanabilir. Örneğin, Xu ve arkadaşları [30], alan-spesifik veri setleriyle yapılan ince ayar (fine-tuning) işlemlerinin, YZ modellerinin bağlamsal uygunluğunu artırabileceğini ve kod üretiminde daha proje-odaklı sonuçlar sunabileceğini göstermiştir. Brown ve arkadaşlarının [6] az örnekle öğrenme üzerine çalışmaları da, proje-spesifik veri setleriyle ince ayar yapılmış modellerin bağlamsal uygunluk sorunlarını azaltabileceğini öne sürmektedir.

Ayrıca, Hendrycks ve arkadaşları [15], karmaşık programlama görevlerinde YZ modellerinin performansını artırmak için daha zengin ve bağlam odaklı veri setlerinin gerektiğini vurgulamıştır. YZ destekli kod üretiminin güvenlik açıkları üzerindeki etkisi, Pearce ve arkadaşlarının [21] vurguladığı üzere, daha derinlemesine incelenmelidir. Mozannar ve arkadaşları [19] tarafından önerilen model sağlamlığı analizleri, Bolt.New gibi araçların üretim ortamlarında güvenilirliğini artırmak için önemli bir araştırma yönü olabilir. Son olarak, YZ araçlarının geliştirici iş akışlarına entegrasyonunun uzun vadeli etkileri, örneğin geliştirici beceri kaybı veya bağımlılık gibi sosyo-teknik boyutlar, ilerideki çalışmalar için önemli bir araştırma alanıdır.

5.4. Sonuç

Bu çalışma, Bolt.New gibi YZ destekli kod üretim araçlarının Python bağlamında insan yazımı kodlarla karşılaştırıldığında, özellikle basit ve orta düzey görevlerde yüksek kalite, düşük hata oranı ve daha iyi stil uyumu sunduğunu ortaya koymaktadır. Karmaşık görevlerde ise YZ, dokümantasyon ve modülerlik açısından avantaj sağlasa da, bağlamsal uygunluk ve algoritmik sezgi gerektiren senaryolarda insan denetimine ihtiyaç duymaktadır. Weisz ve arkadaşları [28], YZ ve insan geliştiriciler arasındaki iş birliğinin, kod üretiminde tamamlayıcı bir rol oynayarak hem verimliliği artırdığını hem de insan uzmanlığının vazgeçilmezliğini koruduğunu vurgulamıştır. Bu bulgular, YZ araçlarının yazılım geliştirme süreçlerinde güçlü bir tamamlayıcı rol oynayabileceğini, ancak insan uzmanlığının vazgeçilmezliğini göstermektedir.

Pratik açıdan, geliştiriciler YZ araçlarını rutin görevlerde verimliliği artırmak için kullanabilir, ancak karmaşık projelerde bağlamsal doğruluk için insan denetimi kritik önemdedir. Teorik açıdan, çalışma YZ destekli kod üretiminin Python'a özgü standartlara uyumunu nesnel metriklerle değerlendirerek literatüre katkı sunmakta ve gelecekteki araştırmalar için bir rehber çerçeve önermektedir. Yazılım mühendisliği ile YZ'nin kesişiminde, otomasyon ve insan yaratıcılığı arasında bir denge kurularak daha verimli, güvenilir ve sürdürülebilir kod üretim süreçleri oluşturulabilir.

Kaynakça

[1] Allamanis, M., Barr, E. T., Devanbu, P., Sutton, C. *A survey of machine learning for big code and naturalness*. ACM Computing Surveys, 51(4), 1–37, 2018. https://doi.org/10,1145/3212695

[2] Beller, M., Bholanath, R., McIntosh, S., Zaidman, A. *Analyzing the state of static analysis: A large-scale evaluation in open source software*. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 470–481, 2016. https://doi.org/10,1109/SANER.2016.13

[3] Bird, J. J., Ekárt, A., Faria, D. R. *Ethical considerations of large language models in code generation: A systematic literature review*. Ethics and Information Technology, 25(3), 1–15, 2023. https://doi.org/10,1007/s10676-023-09684-7

[4] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., MacLeod, G. J., Merritt, M. J. *Characteristics of software quality*. North-Holland Publishing, Amsterdam, Netherlands, 1978.

[5] Bolt.New Team. *Bolt.New: YZ-powered code generation for Python*. https://bolt.new/docs, erişim tarihi: 15 Mayıs 2025.

[6] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Amodei, D. *Language models are few-shot learners*. arXiv preprint arXiv:2005.14165, 2020, https://doi.org/10,48550/arXiv.2005.14165

[7] Buse, R. P., Weimer, W. R. *Learning a metric for code readability*. IEEE Transactions on Software Engineering, 36(4), 546–558, 2010, https://doi.org/10,1109/TSE.2009.71

[8] Carlini, N., Wagner, D. *Adversarial examples are not bugs, they are features*. Advances in Neural Information Processing Systems, 32, 184–194, 2019.

[9] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Zaremba, W. *Evaluating large language models trained on code*. arXiv preprint arXiv:2107.03374, 2021. https://arxiv.org/abs/2107.03374

[10] Dakhel, A. M., Majd, M., Nikanjam, A., Khomh, F., Desmarais, M. C., Jiang, Z. M. J. *GitHub Copilot YZ pair programmer: Asset or liability?* Journal of Systems and Software, 203, 111734, 2023. https://doi.org/10,1016/j.jss.2023.111734

[11] Dijkstra, E. W. *The structure of the "THE"-multiprogramming system*. Communications of the ACM, 11(5), 341–346, 1968. https://doi.org/10,1145/363095.363143

[12] Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., Prather, J. *The robots are coming: Exploring the implications of OpenAI Codex on introductory programming*. In Proceedings of the 24th Australasian Computing Education Conference, pp. 10–19, 2022. https://doi.org/10,1145/3511861.3511863

[13] Goodfellow, I., Bengio, Y., Courville, A. *Deep learning*. MIT Press, 2016.

[14] Halstead, M. H. *Elements of software science*. Elsevier, Amsterdam, Netherlands, 1977.

[15] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Steinhardt, J. *Measuring coding challenge competence with APPS*. Advances in Neural Information Processing Systems, 34, 22214–22226, 2021.

[16] Imai, S. *Is GitHub Copilot a substitute for human pair-programming? An empirical study*. In 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 319–321, 2022. https://doi.org/10,1109/ICSE-Companion55297.2022.9793819

[17] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Vinyals, O. *Competition-level code generation with AlphaCode*. Science, 378(6624), 1092–1097, 2023. https://doi.org/10,1126/science.abq1158

[18] McCabe, T. J. *A complexity measure*. IEEE Transactions on Software Engineering, SE-2(4), 308–320, 1976. https://doi.org/10,1109/TSE.1976.233837

[19] Mozannar, H., Bansal, G., Sontag, D. *Evaluating the robustness of neural code generation models*. arXiv preprint arXiv:2210,12468, 2022. https://doi.org/10,48550/arXiv.2210,12468

[20] Nguyen, T., Li, Y., Le-Cong, T., Nguyen, T. N. *Exploring the impact of code style in Python: An empirical study*. arXiv preprint arXiv:2208.09051, 2022. https://arxiv.org/abs/2208.09051

[21] Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R. *Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions*. In 2022 IEEE Symposium on Security and Privacy (SP), pp. 754–768, 2022. https://doi.org/10,1109/SP46214.2022.9833651

[22] Posnett, D., Hindle, A., Devanbu, P. *A simpler model of software readability*. In Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 73–82, 2011. https://doi.org/10,1145/1985441.1985454

[23] Pylint Development Team. *Pylint: Static code analysis for Python*. https://pylint.readthedocs.io/en/latest/, erişim tarihi: 15 Mayıs 2025.

[24] Python Software Foundation. *PEP 8 – Style guide for Python code*. https://peps.python.org/pep-0008/, 2023.

[25] Scalabrino, S., Linares-Vásquez, M., Oliveto, R., Bavota, G. *A comprehensive model for code readability*. Journal of Software: Evolution and Process, 30(6), e1928, 2018. https://doi.org/10,1002/smr.1928

[26] Sommerville, I. *Software engineering* (10th ed.). Pearson, Boston, MA, 2015.

[27] Vaithilingam, P., Zhang, T., Glassman, E. L. *Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models*. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, pp. 1–14, 2022. https://doi.org/10,1145/3491102.3517665

[28] Weisz, J. D., Muller, M., Houde, S., Richards, J., Ross, S. I., Cabrero, F. *Perfection not required? Human-YZ partnerships in code generation*. In Proceedings of the 2021 AAAI/ACM Conference on YZ, Ethics, and Society, pp. 402–412, 2021. https://doi.org/10,1145/3461702.3462537

[29] Vaswani, A., Shazeer, N., Parmar, N., Uszoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. *Attention is all you need*. Advances in Neural Information Processing Systems, 30, pp. 5998–6008, 2017. https://arxiv.org/abs/1706.03762

[30] Xu, F. F., Vaswani, A., Parmar, N. *Fine-tuning large language models for domain-specific code generation*. arXiv preprint arXiv:2302.01687, 2023.

https://doi.org/10,48550/arXiv.2302.01687

[31] Ziegler, A., Kalliamvakou, E., Simister, A., Sittampalam, G., Li, A., Rice, A., Rifkin, R. *Productivity assessment of neural code completion*. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, pp. 21–29, 2022. https://doi.org/10,1145/3520312.3534864